

《计算概论（B）》（闫宏飞）期末考试参考代码（2017 ~ 2019）



来自 Xzonn 的小站

更新于 2020-11-13 13:18 · 渲染于 2021-01-11 13:05

前言

两年前的我怎么也不会想到我竟然会在大三的时候还在为《计算概论》这门课而头疼。我当年大一的时候上的计概课是马思伟老师主讲的，程序语言是 C。然而我现在在看的是化院开设的计概，主讲老师是闫宏飞，程序语言是 Python。当然，我自己有一些 Python 编程的基础，而且计概 B 讲道理不会特别难，所以我就打开了新世界的大门。

总而言之因为种种原因，我做完了化院开设的《计算概论（B）》（主讲：闫宏飞）的期末考试题。虽然也有别人的作业作为参考，但是我觉得他写得不太详细，因此我自己随手写了一份参考代码出来，也算是造福人民大众了。当然，我的水平肯定不及大佬，让大家见笑了。

本文提供的所有代码均以在课程网站（OpenJudge）上进行过测试，大部分是能通过（Accepted）的，有些题实在通不过（Wrong Answer）的我也放出来了，不怕丢脸，希望大家一起来讨论。

因为我个人不太喜欢在代码里面写注释，因此所有的说明都放在“思路”一段中。

对了，如果你看的是 pdf 版本，那么你复制粘贴代码会很困难（因为有行号而且没有缩进）。建议你自己写一遍，或者扫描第一页的二维码前往网页版。另外，pdf 版本和网页版都是有书签的，方便对题目进行定位。

祝大家期末考试顺利！

来自一条大三的老咸鱼

目录

2017 级期末考试			2018 级期末考试		
题目 ID	标题	页码	题目 ID	标题	页码
A	字符串连接位置	3	A	找魔数	18
B	充实的寒假生活	5	B	24 点	21
C	股票	7	C	2050 年成绩计算	23
D	改卷子	9	D	军备竞赛	28
E	上机考试	12	E	图像的均值滤波	30
F	北大杯台球比赛	15	F	统计套利策略	32
2019 级期末考试					
题目 ID	标题	页码			
A	这一天星期几	35			
B	提取实体	38			
C	图的拉普拉斯矩阵	40			
D	二维矩阵上的卷积运算	42			
E	买学区房	44			
F	因材施教	47			

2017 级期末考试

A. 字符串连接位置

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 16527 提交次数: 539

尝试人数: 152 通过人数: 121 通过率: 80%^①

描述

给定两个字符串 A 和 B，如果 A 的某一后缀恰好和 B 的某一前缀相同，则定义字符串 A 可以连接到 B。

例如，字符串 `xxxxxabc` 可以连接到字符串 `abc*****`，因为

01.	xxxxxabc
02.	abc*****

输入

输入为两行，分别为字符串 A 和 B。

输入保证 A 和 B 是可连接的。

输出

字符串连接的位置，从 0 开始计数。

如果存在多个可连接位置，则输出值最大的位置。

样例输入 1

01.	xxxxxabc
02.	abc*****

样例输出 1

01.	5
-----	---

样例输入 2

01.	a
02.	abcd

样例输出 2

01.	0
-----	---

思路

这道题要求出 A 的后缀与 B 的前缀相同的最小长度。^②

最简单的实现方法就是遍历，依次比较两字符串长为 1 个、2 个、3 个……字符的前后缀是否相等，一旦相等即停止循环（得到最小长度）。而通过利用 Python 的切片特性可以快速获取字符串的前缀、后缀。

如果需要得到最大长度，可反向循环。

^①笔者注：正文所述“提交次数”“尝试人数”“通过人数”“通过率”均为考试数据。参考代码中的“时间”“内存”均为运行在 OpenJudge 上得出的数据。

^②笔者注：由于输出值最大，而输出值 = A 的长度 - 相同前后缀长度，因此需要满足相同前后缀最小。

参考代码

时间: 297 ms 内存: 3244 kB

```
01. a, b = input(), input();
02. n = 0;
03. for i in range(1, min(len(a), len(b)) + 1):
04.     if a[-i:] == b[:i]:
05.         n = i;
06.         break;
07. print(len(a) - n);
```

B. 充实的寒假生活

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 16528 提交次数: 228
尝试人数: 81 通过人数: 31 通过率: 38%

描述

寒假马上就要到了, 龙傲天同学获得了从第 0 天开始到第 60 天结束为期 61 天超长寒假, 他想要尽可能丰富自己的寒假生活。

现提供若干个活动的起止时间, 请计算龙同学这个寒假至多可以参加多少个活动? 注意所参加的活动不能有任何时间上的重叠, 在第 x 天结束的活动和在第 x 天开始的活动不可同时选择。

输入

第一行为整数 n , 代表接下来输入的活动个数 ($n < 10000$) 。

紧接着的 n 行, 每一行都有两个整数, 第一个整数代表活动的开始时间, 第二个整数代表结束时间。

输出

输出至多参加的活动个数。

样例输入

```
01. 5
02. 0 0
03. 1 1
04. 2 2
05. 3 3
06. 4 4
```

样例输出

```
01. 5
```

思路

此题可直接用模拟法求解。先设置变量数组 `days` 用于表明第 i 天是否有空, 然后按顺序比较活动开始时间到结束时间内是否有空, 如有空则可安排活动, 否则不可安排活动。

由于要尽可能多地参加活动, 因此应对全部活动按持续时间排序, 先安排持续时间短的活动。可以证明这种安排方式是最优解。

参考代码

时间: 962 ms 内存: 4532 kB

```
01. n = int(input());
02. l = [];
03. for i in range(n):
04.     l.append([int(i) for i in input().split()]);
05. k = 0;
06. days = [True for i in range(61)];
07. l.sort(key = (lambda x:x[1] - x[0]));
```

```
08.     for i in l:
09.         if all(days[i[0]: i[1] + 1]):
10.             k += 1;
11.             days[i[0]: i[1] + 1] = [False for i in range(i[0], i[1] + 1)];
12.     print(k);
```

C. 股票

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 16529 提交次数: 306
尝试人数: 107 通过人数: 42 通过率: 39%

描述

假设小明一开始有 100 块钱，给出每天的股票价格，小明可以在这些天内先后进行一次买操作和一次卖操作，或者什么也不干。问小明最后最多可以得到多少钱？

注意:

1. 一定要先买后卖。
2. 股票最后一定要卖掉。
3. 数据量较大，如果简单枚举（买入价，卖出价）会超时。

提示: n 天内只能买进一次，卖出一次。并且可以买非整数股。

输入

第一行为天数 n ($1 \leq n \leq 100000$)。

接下来一行有 n 个数 p_i ，为每天的股票价格， $p_i > 0$ 。

输出

最后小明可以得到最多的钱数（小数点后保留两位）。

样例输入 1

01.	5
02.	0.1 0.8 20 0.5 0.01

样例输出 1

01.	20000.00
-----	----------

样例输入 2

01.	6
02.	599 600 301 599 300 301

样例输出 2

01.	199.00
-----	--------

样例输入 3

01.	5
02.	5 4 3 2 1

样例输出 3

01.	100.00
-----	--------

样例输入 4

01.	5
02.	5 4 3 21 1

样例输出 4

01. 700.00

思路

很容易想到, 此题的基本思路是求出在 $i < j$ 的条件下, p_j / p_i 的最大值。

最简单的想法就是直接遍历 $i < j$ 的所有值, 求出所有 p_j / p_i , 然后求最大值。但此方法复杂度为 $O(n^2)$, 在数据量较大时会超时 (我已经试过了)。

网上冲浪之后发现了简化解法, 仅需一层遍历。基本想法是先设置变量 `mi` 用于存放股价的较小值, `rate` 用于存放卖出价与买入价的最大比率。对于每天的股价, 如果其大于等于较小值 `mi`, 则该天买入后无论哪天卖出, 其比率一定小于等于最大比率 `rate`, 因此无需计算直接跳过; 如果其小于较小值 `mi`, 则先判断该天之后的最大股价与该天股价的比值 `r` 是否大于之前的最大比率 `rate`, 若是则将 `mi` 和 `rate` 替换为该天的值, 若否则不进行操作。最后将最大比率 `rate` 乘以初始值 100 元即可。此方法复杂度应为 $O(n)$ 。

参考代码

时间: 2191 ms ^③ 内存: 20332 kB

```
01. n = int(input());
02. l = [float(i) for i in input().split()];
03. mi = l[0];
04. rate = 1.0;
05. for i in range(n):
06.     if l[i] < mi or i == 0:
07.         r = max(l[i + 1: n]) / l[i];
08.         if r > rate:
09.             rate = r;
10.             mi = l[i];
11. print("{0:.2f}".format(rate * 100));
```

^③笔者注: 是的, 运行时间超过了 1000 ms, 但还是通过了。

D. 改卷子

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 16530 提交次数: 299

尝试人数: 118 通过人数: 86 通过率: 73%

描述

闫老师希望将所有学生的卷子按姓名分成两摞，交给两位助教批改。假设每位考生姓名由大写字母组成。

你想出来一个好办法：你给出一个字符串，然后让闫老师将每位同学姓名的每个字母依次与该字符串比较。为了使这个过程更简单，你认为这个字符串应该尽可能的短。

现在有 n 位学生，请你找出这样一个字符串 S ，使得学生姓名小于或等于 S 的有一半，而另一半学生姓名大于 S 。如果有多个字符串满足最短长度，那么就取字母排序最小的那个字符串。

输入

每组数据以 n 开始，代表 n 个学生。（ $n \geq 2$ 且 $n \leq 1000$ 且 n 为偶数）。

接下来 n 行为学生姓名。每个名字都由大写字母组成，长度小于 30 个字母。

输出

针对每组数据，一行输出一个符合条件的最短字符串（大写字母）。

样例输入 1

01.	2
02.	LARHONDA
03.	LARSEN

样例输出 1

01.	LARI
-----	------

样例输入 2

01.	4
02.	SAM
03.	FRED
04.	JOE
05.	MARGARET

样例输出 2

01.	K
-----	---

样例输入 3

01.	2
02.	A
03.	C

样例输出 3

01.	A
-----	---

样例输入 4

01.	2
02.	AYZZ
03.	AZ

样例输出 4

01.	AYZZ
-----	------

思路

这题我是看了别人的代码才发现了自己的错误……丢脸……

显然此题需要先对名字排序，然后选取最中间的两个字符 A、B 进行比较^④。基本思路如下：

首先先将 A、B 的最长相同前缀提取出来，这部分对最终的结果无影响。

如果此时 A 无剩余字符或仅剩余 1 个字符，由于必有 $A < B$ ，则 A 本身即是满足条件的最短字符串，返回 A 即可。例如：A & AB，返回 A；AA & AB，返回 AA。

设 A 剩余字符为 α ，B 剩余字符为 β 。若 $\beta_1 - \alpha_1 > 1$ (α_1 表示 α 的第 1 个字符的 ASCII 码，后同)，则只需返回最长前缀后接 $(\alpha_1 + 1)$ 即可。此时由于 α_1 不可能为 Z，因此无需考虑超出范围。例如：AB & C，返回 B。

若 $\beta_1 - \alpha_1 = 1$ 且 B 有大于等于 2 个剩余字符，此时同样返回前缀后接 $(\alpha_1 + 1)$ 。同样， α_1 不可能为 Z。例如：AB & BA，返回 B。^⑤

若 $\beta_1 - \alpha_1 = 1$ ，此时 α_1 可能为 Z。需要按位置顺序依次检查字符是否为 Z，并在第一个不为 Z 的字符上 +1（如果此字符为结尾字符，则直接返回该字符）。例如：AZAA & B，返回 AZB；AZA & B，返回 AZA。

参考代码

时间：875 ms 内存：3616 kB

```
01. def main(n, nameList):
02.     nameList.sort();
03.     a, b = nameList[n // 2 - 1], nameList[n // 2];
04.     i = 0;
05.     while i < min(len(a), len(b)):
06.         if a[:i + 1] != b[:i + 1]:
07.             break;
08.         i += 1;
09.     if len(a) <= i + 1:
10.         return a;
11.     elif (ord(b[i]) - ord(a[i]) > 1) or len(b) > i + 1:
12.         return a[:i] + chr(ord(a[i]) + 1);
13.     else:
14.         for i in range(i + 1, len(a) - 1):
15.             if a[i] != 'Z':
16.                 return a[:i] + chr(ord(a[i]) + 1);
17.         return a;
```

^④笔者注：此处比较方法是字典序比较。其基本思路与字典的单词排序顺序类似，即先比较两字符串第一个字符，若相同则比较下一个字符。若出现不同，则字符较小者靠前。例如：ABC < ABD。若某字符串先结束，则较短者靠前。例如：ABC < ABCD。

^⑤笔者注：我第一次做这道题的时候就是没有考虑到这种情况，然后 Debug 好久都没发现……

```
18.  
19.     n = int(input());  
20.     nameList = [];  
21.     for i in range(n):  
22.         nameList.append(input());  
23.     print(main(n, nameList));
```

E. 上机考试

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 16531 提交次数: 165
尝试人数: 47 通过人数: 12 通过率: 26%

描述

一个班的同学在 m 行 \times n 列的机房考试，给出学生的座位分布和每个人的做题情况，统计做题情况与周围（上下左右）任一同学相同的学生人数。

另外，由于考试的优秀率不能超过 40%，请统计这个班的优秀人数（可能为 0，相同分数的同学必须要么全是优秀，要么全不是优秀）。

提示: $m \times n$ 行做题情况可能为空。因为如果所有学生做题过程中，一直没有提交，则空。⑥

输入

第一行为两个整数， m 和 n 。

接下来 m 行每行 n 个整数，代表对应学生的编号，编号各不相同，范围由 0 到 $m \times n - 1$ 。

接下来 $m \times n$ 行，顺序给出编号由 0 到 $n - 1$ 的同学的做题情况，1 代表做对，0 代表做错。

输出

两个整数，以空格分隔，分别代表 “与周围任一同学做题情况相同的学生人数” 和 “班级优秀的人数”。

样例输入 1

01.	2 5
02.	0 1 2 3 4
03.	5 6 7 8 9
04.	1 1 1
05.	1 0 1
06.	0 0 0
07.	0 0 1
08.	0 0 0
09.	1 1 1
10.	1 1 1
11.	1 1 1
12.	1 1 1
13.	1 1 1

样例输出 1

01.	6 0
-----	-----

解释:

编号为 0 5 6 7 8 9 的同学做题情况与周围同学相同，因此第一个整数是 6。

全对的同学人数已经超过了 40%，因此优秀的人数为 0。

⑥笔者注：我个人没有看懂这句话的含义，在编写程序时也没有考虑这种情况，提交之后也通过了。

样例输入 2

01.	1 3
02.	1 0 2
03.	0 1 0 0
04.	0 0 0 0
05.	0 0 0 0

样例输出 2

01.	0 1
-----	-----

解释：并不存在与相邻同学做题情况相同的同学，并且做对一题的同学比例小于 40%，因此有一人优秀。

样例输入 3

01.	2
02.	A
03.	C

样例输出 3

01.	A
-----	---

样例输入 4

01.	2
02.	AYZZ
03.	AZ

样例输出 4

01.	AYZZ
-----	------

思路

这题思路倒是不难，用模拟法求解就可以。通过率低可能是因为考试写不完了。“做题相同人数”和“优秀人数”分别求解。

先看“优秀人数”。先设置变量字典 `scores` 用于存放各成绩的人数，`count` 用于存放优秀人数，`maxCount` 用于存放最大优秀人数。先遍历所有学生，求出总成绩，然后存放入 `scores` 对应的值中。然后对成绩倒序排序后遍历，如果某成绩的人数加上已有优秀人数大于最大优秀人数，则结束遍历。

再看“做题相同人数”。我一直认为会有简化算法，但是想了半天还是用了遍历思想。先设置列表 `same` 用于存放某编号学生是否与周围人做题相同，然后用两个二重循环，一个用于横向两两比较，另一个用于竖向两两比较。若比较结果为两人相同，则将 `same` 对应的值设为 `True`。最后对 `same` 中 `True` 的数目进行计数即可。

参考代码

时间：749 ms 内存：3480 kB

01.	<code>m, n = (int(i) for i in input().split());</code>
02.	<code>ids = [];</code>
03.	<code>for i in range(m):</code>
04.	<code>ids.append([int(i) for i in input().split()]);</code>

```
05. results = [None for i in range(m * n)];
06. scores = {};
07. for i in range(m * n):
08.     results[i] = [int(i) for i in input().split()];
09.     s = sum(results[i]);
10.     if s in scores:
11.         scores[s] += 1;
12.     else:
13.         scores[s] = 1;
14. count = 0;
15. maxCount = m * n * 0.4;
16. for i in sorted(scores.keys(), reverse = True):
17.     if (count + scores[i]) > maxCount:
18.         break;
19.     else:
20.         count += scores[i];
21. same = [False for i in range(m * n)];
22. for i in range(m):
23.     for j in range(n - 1):
24.         if (results[ids[i][j]] == results[ids[i][j + 1]]):
25.             same[ids[i][j]] = True;
26.             same[ids[i][j + 1]] = True;
27. for i in range(m - 1):
28.     for j in range(n):
29.         if (results[ids[i][j]] == results[ids[i + 1][j]]):
30.             same[ids[i][j]] = True;
31.             same[ids[i + 1][j]] = True;
32. print(same.count(True), count);
```

F. 北大杯台球比赛

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 16532 提交次数: 81
 尝试人数: 23 通过人数: 11 通过率: 48%

描述

北大杯台球比赛进入白热化的黑八大战环节，台球桌尺寸为 16×5 ^⑦，以台球桌左上角建立坐标系如下图所示：

01.	0—1—2—3—4—5—6—7—8—...—16—	x轴
02.		
03.	1—.—.—.—.—.—.—.—.—.—.—.	
04.		
05.	2—.—.—.—.—.—.—.—.—.—.—.	
06.		
07.	3—.—.—.—.—.—.—.—.—.—.—.	
08.		
09.	4—.—.—.—.—.—.—.—.—.—.—.	
10.		
11.	5—.—.—.—.—.—.—.—.—.—.—.	
12.		
13.		
14.	y轴	

其中 (0, 0)、(8, 0)、(16, 0)、(0, 5)、(8, 5)、(16, 5) 是台球桌上的六个入袋口，台球只有运动到这六个网格顶点时才能进球。

现在已知台球桌上仅剩下一个白色母球和一个黑球，并且知道他们的 x, y 坐标（均为整数）。击球时只能击打白色母球，且击球方向只有左上 (-1, -1)、左下 (-1, 1)、右上 (1, -1)、右下 (1, 1) 四种方向。

击球后白色母球会沿击球方向运动，若碰到球桌壁则会发生反弹，若碰到黑球则会发生完全弹性碰撞导致动能完全传递（即白球静止，黑球获得白球的速度继续运动）。球向以上 4 个方向移动一次就会消耗一单位能量。请计算最后的胜负情况。

输入

输入为四行。

第一行为白色母球的坐标。

第二行为黑球的坐标。

第三行为击球方向。

第四行为击球力度，即施加给白球的初始动能是多少单位。

输入保证两球的坐标落在球桌内，不会在球壁上，且不重复，保证击球方向为上述四方向之一，保证击球能量 ≥ 0 。

输出

输出为一行。

^⑦ 笔者注：原文如此。此说法似乎不妥，坐标编号为 (0 ~ 16, 0 ~ 5)，尺寸应为 17×6 。

假如白球入库输出-1，黑球入库则输出 1，无球入库则输出 0。

样例输入 1

01.	1 1
02.	2 2
03.	-1 -1
04.	10

样例输出 1

01.	-1
-----	----

样例输入 2

01.	2 2
02.	1 1
03.	-1 -1
04.	10

样例输出 2

01.	1
-----	---

样例输入 3

01.	2 2
02.	1 1
03.	-1 -1
04.	0

样例输出 3

01.	0
-----	---

思路

这题我的第一想法是找出运动轨迹然后直接计算，但是考虑到这是道考试题，大概也没时间做花里胡哨的东西，所以还是直接模拟法求解吧。

为了简化代码，我把“无球”定义为 `0`，“白球”定义为 `1`，“黑球”定义为 `2`。先设置变量数组 `pos` 用于存放白球和黑球的位置，`direc` 用于存放正在运动的球的运动方向，`force` 用于存放球的剩余能量，`now` 用于存放正在运动的球，`map` 用于存放地图各点的状态，`holes` 用于存放所有球洞的位置。开始时先将 `map` 中白球和黑球的点分别设置为 `1` 和 `2`（代表这两点上有白球和黑球），其他位置为 `0`（代表这些点上没有球）。

在保证能量 `force > 0` 的情况下循环：先计算正在运动的球的位置 `newPos = pos[now] + direc`^⑧。先判断特殊情况：

- 若新位置在洞中，则直接输出掉入洞中的球（白球输出 `-1`，黑球输出 `1`），结束程序。
- 若新位置上有球，则将正在运动的球改为新位置上的球，由于完全弹性碰撞没有能量损失，因此 `force` 不变，继续循环。

⑧笔者注：此处为简化写法，实际上应对 `x` 和 `y` 坐标分别相加。

- 若新位置为桌壁，则改变运动方向（与左右壁碰撞改变 x 方向，与上下壁碰撞改变 y 方向）。

判断特殊情况后，将 `map` 上一个位置的数值设为 `0`，将现在的位置 `pos[now]` 设为新位置 `newPos`，将 `map` 新位置的数值设为正在运动的球 `now`。然后将能量 `force` - 1，继续循环。

如果循环结束时没有返回值（即没有球入洞），则返回 `0`。

参考代码

时间: 403 ms 内存: 3396 kB

```

01. def main():
02.     pos = [None, [int(i) for i in input().split()], [int(i) for i in input().split()]];
03.     direc = [int(i) for i in input().split()];
04.     force = int(input());
05.     now = 1;
06.     map = [[0 for i in range(6)] for j in range(17)];
07.     holes = [(0, 0), (8, 0), (16, 0), (0, 5), (8, 5), (16, 5)];
08.
09.     map[pos[1][0]][pos[1][1]] = 1;
10.     map[pos[2][0]][pos[2][1]] = 2;
11.     while force:
12.         newPos = (pos[now][0] + direc[0], pos[now][1] + direc[1]);
13.         if newPos in holes:
14.             return -1 if now == 1 else 1;
15.         elif map[newPos[0]][newPos[1]] != 0:
16.             now = map[newPos[0]][newPos[1]];
17.             continue;
18.         elif newPos[0] in (0, 16):
19.             direc[0] = -direc[0];
20.         elif newPos[1] in (0, 5):
21.             direc[1] = -direc[1];
22.         map[pos[now][0]][pos[now][1]] = 0;
23.         pos[now] = newPos;
24.         map[pos[now][0]][pos[now][1]] = now;
25.         force -= 1;
26.     return 0;
27.
28. print(main());

```

2018 级期末考试

A. 找魔数

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 18224 提交次数: 490
尝试人数: 133 通过人数: 105 通过率: 79%

描述

一个数如果能表示为两个完全平方数的和（两个正整数的平方和，如： $2^2 + 4^2 = 20$ ， $1^2 + 4^2 = 17$ ），则称为魔数。要求你在一堆数字中找出魔数，并且分别以二进制、八进制、十六进制输出这个数字。

输入

第一行 1 个整数，代表 m 个数字（ $1 \leq m \leq 100$ ）。
接下来 1 行，为 m 个整数， x_i （ $1 \leq x_i \leq 1000$ ）。

输出

每个魔数 1 行，以空格间隔输出该魔数的二进制、八进制、十六进制。
（注：如果出现 abcdef 等字母，都按小写输出）。

样例输入

01.	4
02.	3 9 20 17

样例输出

01.	0b10100 0o24 0x14
02.	0b10001 0o21 0x11

思路

“魔数”其实就是两个完全平方数的和，由于 $1 \leq x_i \leq 1000$ ，因此最暴力的方法实际上就是把 1 ~ 1000 以内的所有“魔数”求出来，然后从输入的数字里面把“魔数”挑选出来，再进行后续输出操作。而输出二进制、八进制、十六进制则直接调用 Python 内置函数 `bin()`、`oct()`、`hex()` 即可。

至于求魔数的方法，可以单独写一段 Python 代码求，也可以直接用 Excel 求。由于 $31^2 = 961$ ， $32^2 = 1024$ ，所以只需要求出 1 ~ 31 的平方并两两加和，去重后得到列表 `data`。

Python 代码见下：

01.	<code>magicNumbers = [];</code>
02.	<code>for i in range(1, 32):</code>
03.	<code> for j in range(i, 32):</code>
04.	<code> magicNumbers.append(i * i + j * j);</code>
05.	<code>magicNumbers = sorted(list(set(magicNumbers)));</code>
06.	<code>print(magicNumbers);</code>

Excel 求法见图 1。图中 A 列和 1 行是 1 ~ 31 的数列，B 列和 2 行是其平方，C3 ~ AG33 单元格是平方和。然后复制 C3 ~ AG33 单元格中的内容直接粘贴到 Python 中作为长字符串（用 `''` 包含），然后用 `split` 方法分隔成单独的数字即可。

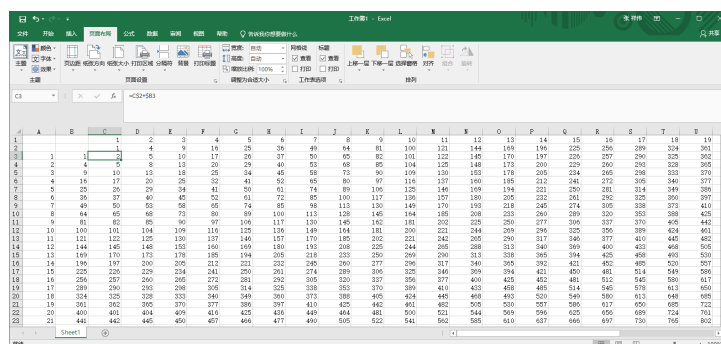


图 1 用 Excel 求“魔数”

暴力法简单粗暴好想，但是在数据量大的时候会消耗大量内存，同时在查表的时候也会消耗大量时间。另一种思想是对于给定的数，直接判断该数是否为“魔数”。数学上或许可以证明了两个完全平方数的和的性质，但是考试时间紧，数学证明不太现实。因此这里同样给出遍历判断的 Python 代码：

```
01. from math import sqrt
02.
03. magicNumbers = {};
04. def isMagicNumber(x):
05.     global magicNumbers;
06.     if x in magicNumbers:
07.         return magicNumbers[x];
08.     else:
09.         maxRoot = int(sqrt(x));
10.         for i in range(1, maxRoot + 1):
11.             for j in range(i, maxRoot + 1):
12.                 if x == i * i + j * j:
13.                     magicNumbers[x] = True;
14.                     return True;
15.         magicNumbers[x] = False;
16.     return False;
```

上述代码中利用了“缓存”的思想，即维护一个字典 `magicNumbers`，如果一个数 `x` 已被判断过，则直接输出 `magicNumbers[x]`。否则，求出 `x` 的平方根的下取整值 `maxRoot`，进行二重循环 ($1 \sim \text{maxRoot}$ ， $i \sim \text{maxRoot}$)，判断 `x` 是否等于两数平方和。若是则保存 `magicNumbers[x] = True` 并返回 `True`；若循环结束仍未返回 `True` 则保存 `magicNumbers[x] = False` 并返回 `False`。

下述参考代码中直接使用了暴力法，直接将计算好的“魔数”保存为数组 `data` 减少运算量。在判断某数是否为“魔数”时，利用了 Python 内置的 `filter()` 函数和 lambda 表达式的写法。

如果想用判断法，可参考第 C 题的写法。

参考代码

时间：709 ms 内存：3604 kB

```
01. data = [2, 5, 8, 10, 13, 17, 18, 20, 25, 26, 29, 32, 34, 37, 40, 41, 45, 50, 52, 53, 58, 61, 65,
68, 72, 73, 74, 80, 82, 85, 89, 90, 97, 98, 100, 101, 104, 106, 109, 113, 116, 117, 122, 125,
128, 130, 136, 137, 145, 146, 148, 149, 153, 157, 160, 162, 164, 169, 170, 173, 178, 180, 181,
185, 193, 194, 197, 200, 202, 205, 208, 212, 218, 221, 225, 226, 229, 232, 233, 234, 241, 242,
244, 245, 250, 257, 260, 261, 265, 269, 272, 274, 277, 281, 288, 289, 290, 292, 293, 296, 298,
```

	305, 306, 313, 314, 317, 320, 325, 328, 333, 337, 338, 340, 346, 349, 353, 356, 360, 362, 365, 369, 370, 373, 377, 386, 388, 389, 392, 394, 397, 400, 401, 404, 405, 409, 410, 416, 421, 424, 425, 433, 436, 442, 445, 449, 450, 452, 457, 458, 461, 464, 466, 468, 477, 481, 482, 485, 488, 490, 493, 500, 505, 509, 512, 514, 520, 521, 522, 530, 533, 538, 541, 544, 545, 548, 549, 554, 557, 562, 565, 569, 577, 578, 580, 584, 585, 586, 592, 593, 596, 601, 605, 610, 612, 613, 617, 625, 626, 628, 629, 634, 637, 640, 641, 648, 650, 653, 656, 657, 661, 666, 673, 674, 676, 677, 680, 685, 689, 692, 697, 698, 701, 706, 709, 712, 720, 722, 724, 725, 730, 733, 738, 740, 745, 746, 754, 757, 761, 765, 769, 772, 773, 776, 778, 785, 788, 793, 794, 797, 800, 801, 802, 808, 809, 810, 818, 820, 821, 829, 832, 833, 841, 842, 845, 848, 850, 853, 857, 865, 866, 872, 873, 877, 881, 882, 884, 890, 898, 900, 901, 904, 905, 909, 914, 916, 922, 925, 928, 929, 932, 936, 937, 941, 949, 953, 954, 962, 964, 965, 968, 970, 976, 977, 980, 981, 985, 986, 997, 1000]
02.	n = int(input());
03.	numbers = [int(i) for i in input().split()];
04.	magicNumbers = filter(lambda x: x in data, numbers);
05.	for i in magicNumbers:
06.	print(" ".join([bin(i), oct(i), hex(i)]));

B. 24 点

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 18223 提交次数: 254

尝试人数. 139 通过人数. 123 通过率. 88%

描述

给定 4 个整数，判断是否能只用加减运算（即在 4 个数中间插入 3 个+或-符号，可以调换数字顺序），使得运算结果为 24。

输入

第一行 1 个整数, 代表 m 组数据 ($1 \leq m \leq 100$)

接下来 m 行, 每行为 4 个整数, x_i ($1 \leq x_i \leq 10^{100}$), 注意整数可能很大。

输出

共 m 行, 每行为 YES 或者 NO, 代表是否能凑成 24 点。

[illegible]

样例输入

01.	4
02.	6 6 6 6
03.	3 25 1 1
04.	10000000000000000000000000000000 10000000000000000000000000000000 10000000000000000000000000000000 1000000000000000000000000000000024
05.	2 3 4 5

样例输出

01.	YES
02.	YES
03.	YES
04.	NO

思路

虽然题目中有说“可以调换数字顺序”，但是因为题目限定了加减运算，因此只需要考虑四个数前面的符号是“+”还是“-”即可。如果遍历的话，理论上要运算 $2^4 = 16$ 次，但是考虑到形如“++++”和“----”的两组符号得到的结果互为相反数（即如果“++++”得到24，则“----”得到-24），因此只需要运算8次，比较运算结果是否等于24或-24即可。

参考代码中利用了 Python 内置的 `map()` 函数简化代码量，也可以用 `for` 循环代替，且 `for` 循环可以在得到 24 或 -24 后立刻跳出循环，理论上可以节省时间。

参考代码

时间. 750 ms 内存. 3516 kB

```
01. matrix = ((1, 1, 1, 1),
02.           (1, 1, 1, -1),
03.           (1, 1, -1, 1),
```

```
04.         (1, 1, -1, -1),
05.         (1, -1, 1, 1),
06.         (1, -1, 1, -1),
07.         (1, -1, -1, 1),
08.         (1, -1, -1, -1));
09.     n = int(input());
10.     for i in range(n):
11.         numbers = [int(i) for i in input().split()];
12.         def calc(a):
13.             return numbers[0] * a[0] + numbers[1] * a[1] + numbers[2] * a[2] + numbers[3] * a[3];
14.         results = list(map(calc, matrix));
15.         print("YES" if (24 in results or -24 in results) else "NO");
```

C. 2050 年成绩计算

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 18176 提交次数: 442
 尝试人数: 112 通过人数: 40 通过率: 36%

描述

随着人工智能技术和基因编辑技术的普及，2050 年的学生成绩计算与目前 2018 年的规则很不一样。课程成绩要求是 **t-prime**，才是有效成绩，否则按照零分计算。

现在需要统计每位学生在一个学期中，所有有效成绩课程的平均分，需要你的帮忙。

(素数是指除了 1 和它本身以外, 不能被任何整数整除的数。类似地, 如果整数 t 恰好有且仅有三个不同的正除数^⑨, 我们将称为它为 **t-prime**。)

输入

第一行为两个整数, m 个学生 ($1 \leq m \leq 2000$), n 门课 ($1 \leq n \leq 100$, 每个学生选课数可以不一样)。

接下来 m 行, 每行代表学生选课获得成绩。成绩由空格分隔, 成绩是整数 x_i ($1 \leq x_i \leq 10^8$)。

输出

共 m 行, 每行对应一位学生有效成绩课程的平均分 (小数点后保留两位)。如果该生所有选课有效成绩是零分, 则输出 0。

对如下样例, 第一个学生所有成绩都不是 **t-prime**, 因此输出 0, 注意这里不保留两位小数。

第二位学生只有 4 分是有效分数, 故输出 $4 / 3 = 1.33$ 。

第三位学生只有两门课有成绩, 4 和 25 均为 **t-prime**, 因此输出平均值 14.50。

样例输入

01.	3 3
02.	100 120 2
03.	3 4 5
04.	4 25

样例输出

01.	0
02.	1.33
03.	14.50

思路

t-prime 看似复杂, 其实很好理解。除了 1 之外的数至少都有 1 和它本身作为因数, 而 **t-prime** 有且仅有 3 个因数, 因此第 3 个因数应为其平方根, 且该平方根为质数。因此判断一个数是否为 **t-prime**, 只需判断该数是否为完全平方数, 以及该数的平方根是否为质数。

参考第 A 题, 很容易想到两种解法, 即暴力法和判断法。暴力法的思想是求出 $1 \sim 10^8$ 范围内的所有 **t-prime**, 进而想到求出 $1 \sim 10^4$ 范围内的所有质数。而质数的计算方法较简单, 只需计算该数是否能被 $2 \sim$ 该数平方根的范围内的整数整除即可。参考代码如下:

^⑨笔者注: 原文如此。个人认为应该称其为“因数”。

```

01.  from math import sqrt
02.
03.  prime = [True for i in range(10000)];
04.  prime[0:1] = [False, False];
05.  for i in range(1, 10000):
06.      for j in range(2, int(sqrt(i)) + 1):
07.          if i % j == 0:
08.              prime[i] = False;
09.              break;
10.  primes = [];
11.  for i in range(1, 10000):
12.      if prime[i]:
13.          primes.append(i);
14.  tPrimes = [i * i for i in primes];
15.  print(tPrimes);

```

输出如下:

```

01.  [4, 9, 25, 49, 121, 169, 289, 361, 529, 841, 961, 1369, 1681, 1849, 2209, 2809, 3481, 3721, 4489,
    5041, 5329, 6241, 6889, 7921, 9409, 10201, 10609, 11449, 11881, 12769, 16129, 17161, 18769,
    19321, 22201, 22801, 24649, 26569, 27889, 29929, 32041, 32761, 36481, 37249, 38809, 39601,
    44521, 49729, 51529, 52441, 54289, 57121, 58081, 63001, 66049, 69169, 72361, 73441, 76729,
    78961, 80089, 85849, 94249, 96721, 97969, 100489, 109561, 113569, 120409, 121801, 124609,
    128881, 134689, 139129, 143641, 146689, 151321, 157609, 160801, 167281, 175561, 177241, 185761,
    187489, 192721, 196249, 201601, 208849, 212521, 214369, 218089, 229441, 237169, 241081, 249001,
    253009, 259081, 271441, 273529, 292681, 299209, 310249, 316969, 323761, 326041, 332929, 344569,
    351649, 358801, 361201, 368449, 375769, 380689, 383161, 398161, 410881, 413449, 418609, 426409,
    434281, 436921, 452929, 458329, 466489, 477481, 491401, 502681, 516961, 528529, 537289, 546121,
    552049, 564001, 573049, 579121, 591361, 597529, 619369, 635209, 654481, 657721, 674041, 677329,
    683929, 687241, 703921, 727609, 734449, 737881, 744769, 769129, 776161, 779689, 786769, 822649,
    829921, 844561, 863041, 877969, 885481, 896809, 908209, 935089, 942841, 954529, 966289, 982081,
    994009, 1018081, 1026169, 1038361, 1042441, 1062961, 1067089, 1079521, 1100401, 1104601,
    1125721, 1129969, 1142761, 1181569, 1190281, 1194649, 1203409, 1216609, 1229881, 1247689,
    1261129, 1274641, 1324801, 1329409, 1352569, 1371241, 1394761, 1408969, 1423249, 1442401,
    1471369, 1481089, 1495729, 1510441, 1515361, 1530169, 1560001, 1585081, 1630729, 1635841,
    1646089, 1661521, 1666681, 1682209, 1692601, 1697809, 1708249, 1739761, 1745041, 1760929,
    1852321, 1868689, 1885129, 1907161, 1957201, 1985281, 2024929, 2036329, 2042041, 2053489,
    2070721, 2093809, 2105401, 2111209, 2128681, 2163841, 2193361, 2199289, 2211169, 2217121,
    2229049, 2247001, 2283121, 2319529, 2343961, 2380849, 2399401, 2411809, 2430481, 2455489,
    2468041, 2493241, 2505889, 2550409, 2563201, 2582449, 2588881, 2601769, 2621161, 2627641,
    2647129, 2679769, 2745649, 2765569, 2778889, 2785561, 2866249, 2879809, 2886601, 2920681,
    2961841, 2968729, 3003289, 3031081, 3052009, 3073009, 3094081, 3157729, 3179089, 3193369,
    3200521, 3243601, 3279721, 3323329, 3352561, 3411409, 3463321, 3485689, 3500641, 3508129,
    3523129, 3530641, 3568321, 3613801, 3636649, 3659569, 3728761, 3736489, 3798601, 3806401,
    3892729, 3916441, 3948169, 3972049, 3988009, 3996001, 4012009, 4044121, 4068289, 4108729,
    4116841, 4157521, 4214809, 4255969, 4280761, 4330561, 4338889, 4355569, 4363921, 4405801,
    4456321, 4464769, 4532641, 4541161, 4566769, 4583881, 4592449, 4635409, 4669921, 4748041,
    4853209, 4870849, 4897369, 4932841, 5004169, 5013121, 5031049, 5067001, 5139289, 5148361,

```


5166529, 5202961, 5230369, 5257849, 5276209, 5331481, 5340721, 5442889, 5470921, 5480281, 5508409, 5527201, 5555449, 5621641, 5650129, 5669161, 5678689, 5707321, 5726449, 5755201, 5812921, 5841889, 5870929, 5938969, 5958481, 5987809, 6046681, 6086089, 6115729, 6135529, 6265009, 6355441, 6405961, 6446521, 6466849, 6497401, 6507601, 6538249, 6651241, 6713281, 6723649, 6806881, 6848689, 6869641, 6932689, 7006609, 7059649, 7070281, 7091569, 7134241, 7166329, 7198489, 7219969, 7230721, 7252249, 7284601, 7327849, 7349521, 7360369, 7392961, 7447441, 7458361, 7513081, 7557001, 7579009, 7656289, 7711729, 7778521, 7789681, 7823209, 7845601, 7856809, 7946761, 8025889, 8048569, 8082649, 8128201, 8162449, 8185321, 8288641, 8334769, 8392609, 8427409, 8462281, 8508889, 8567329, 8637721, 8720209, 8743849, 8779369, 8814961, 8826841, 8994001, 9006001, 9066121, 9114361, 9138529, 9223369, 9247681, 9296401, 9369721, 9406489, 9480241, 9504889, 9541921, 9665881, 9728161, 9740641, 9840769, 10004569, 10029889, 10042561, 10118761, 10156969, 10182481, 10259209, 10297681, 10349089, 10374841, 10426441, 10569001, 10582009, 10608049, 10621081, 10699441, 10883401, 10896601, 10936249, 10975969, 11015761, 11042329, 11082241, 11095561, 11175649, 11202409, 11282881, 11296321, 11363641, 11377129, 11485321, 11498881, 11607649, 11648569, 11785489, 11895601, 11950849, 11978521, 11992369, 12020089, 12033961, 12187081, 12243001, 12327121, 12369289, 12439729, 12453841, 12482089, 12524521, 12538681, 12581209, 12652249, 12666481, 12752041, 12823561, 12837889, 12909649, 13010449, 13053769, 13082689, 13126129, 13184161, 13227769, 13271449, 13388281, 13476241, 13490929, 13520329, 13623481, 13667809, 13697401, 13756681, 13830961, 13890529, 13935289, 13980121, 14145121, 14190289, 14205361, 14280841, 14386849, 14417209, 14462809, 14600041, 14615329, 14691889, 14799409, 14830201, 14845609, 14922769, 15031129, 15062161, 15124321, 15264649, 15295921, 15342889, 15358561, 15389929, 15437041, 15452761, 15547249, 15578809, 15737089, 15912121, 16008001, 16024009, 16056049, 16104169, 16152361, 16168441, 16216729, 16394401, 16410601, 16459249, 16589329, 16638241, 16736281, 16752649, 16801801, 16900321, 17032129, 17048641, 17081689, 17131321, 17247409, 17280649, 17297281, 17447329, 17648401, 17732521, 17783089, 17799961, 17884441, 17901361, 17986081, 18003049, 18088009, 18139081, 18156121, 18241441, 18258529, 18344089, 18395521, 18464209, 18722929, 18809569, 18826921, 18913801, 18983449, 19035769, 19123129, 19280881, 19333609, 19439281, 19545241, 19562929, 19722481, 19775809, 19811401, 19864849, 19918369, 20079361, 20097289, 20187049, 20313049, 20367169, 20403289, 20421361, 20457529, 20675209, 20693401, 20802721, 20857489, 21003889, 21077281, 21132409, 21187609, 21353641, 21501769, 21520321, 21557449, 21613201, 21631801, 21687649, 21743569, 21836929, 21893041, 22005481, 22118209, 22287841, 22306729, 22363441, 22401289, 22572001, 22648081, 22877089, 22915369, 22934521, 22972849, 23030401, 23049601, 23164969, 23203489, 23338561, 23629321, 23726641, 23785129, 23902321, 24039409, 24098281, 24196561, 24314761, 24334489, 24373969, 24433249, 24512401, 24571849, 24671089, 24690961, 24730729, 24870169, 24930049, 24990001, 25030009, 25090081, 25110121, 25210441, 25230529, 25391521, 25512601, 25593481, 25775929, 25816561, 25877569, 25999801, 26020201, 26081449, 26142769, 26204161, 26491609, 26553409, 26697889, 26739241, 26822041, 26925721, 27008809, 27133681, 27321529, 27363361, 27384289, 27426169, 27678121, 27804529, 27867841, 27888961, 28058209, 28121809, 28185481, 28334329, 28440889, 28590409, 28633201, 28955161, 29019769, 29084449, 29149201, 29235649, 29300569, 29343889, 29365561, 29495761, 29560969, 29604481, 29626249, 29691601, 29931841, 29997529, 30019441, 30063289, 30261001, 30283009, 30327049, 30459361, 30481441, 30547729, 30591961, 30880249, 30946969, 31013761, 31058329, 31147561, 31259281, 31618129, 31798321, 31820881, 31888609, 31933801, 31956409, 32001649, 32024281, 32137561, 32296489, 32364721, 32410249, 32501401, 32615521, 32684089, 32913169, 32959081, 32982049, 33051001, 33396841, 33443089, 33535681, 33651601, 33721249,

33790969, 33884041, 33953929, 34093921, 34140649, 34210801, 34234201, 34304449, 34351321,
 34421689, 34445161, 34562641, 34586161, 34774609, 34845409, 35081929, 35129329, 35271721,
 35438209, 35772361, 35844169, 36084049, 36132121, 36348841, 36445369, 36517849, 36566209,
 36638809, 36808489, 36881329, 36954241, 37075921, 37100281, 37222201, 37368769, 37466641,
 37589161, 37613689, 37736449, 37834801, 37982569, 38105929, 38402809, 38427601, 38477209,
 38576521, 38651089, 38700841, 38800441, 39025009, 39150049, 39225169, 39300361, 39325441,
 39400729, 39526369, 39677401, 39702601, 39828721, 39904489, 39980329, 40056241, 40157569,
 40233649, 40360609, 40436881, 40462321, 40538689, 40615129, 40691641, 40819321, 40921609,
 41229241, 41306329, 41589601, 41615401, 41847961, 41899729, 42003361, 42133081, 42523441,
 42627841, 42863209, 42915601, 42941809, 43072969, 43151761, 43178041, 43256929, 43309561,
 43546801, 43652449, 43811161, 44049769, 44262409, 44342281, 44368921, 44528929, 44609041,
 44742721, 44769481, 44903401, 44930209, 45010681, 45144961, 45333289, 45387169, 45711121,
 45738169, 45954841, 45981961, 46117681, 46144849, 46280809, 46553329, 46607929, 46635241,
 46689889, 46799281, 47018449, 47100769, 47183161, 47210641, 47375689, 47596201, 47706649,
 47761921, 47844889, 48260809, 48288601, 48427681, 48455521, 48539089, 48594841, 48678529,
 48762289, 48874081, 48958009, 49014001, 49182169, 49266361, 49378729, 49547521, 49603849,
 49801249, 49970761, 50112241, 50452609, 50537881, 50708641, 50794129, 50822641, 51136801,
 51251281, 51509329, 51652969, 51739249, 51940849, 51998521, 52027369, 52113961, 52258441,
 52374169, 52461049, 52519009, 52606009, 53042089, 53246209, 53392249, 53421481, 53597041,
 53743561, 53772889, 54007801, 54037201, 54302161, 54656449, 54922921, 55011889, 55249489,
 55517401, 55606849, 55636681, 55905529, 55965361, 56055169, 56085121, 56235001, 56355049,
 56505289, 56595529, 56685841, 56806369, 56866681, 56957209, 56987401, 57138481, 57168721,
 57350329, 57410929, 57501889, 57592921, 57623281, 57805609, 57866449, 58079641, 58354321,
 58415449, 58507201, 58813561, 58874929, 58997761, 59089969, 59151481, 59274601, 59336209,
 59552089, 59644729, 59706529, 59923081, 60109009, 60171049, 60202081, 60668521, 60730849,
 61105489, 61199329, 61293241, 61481281, 61669609, 61889689, 61984129, 62047129, 62078641,
 62141689, 62425801, 62520649, 62710561, 62837329, 62932489, 62995969, 63186601, 63218401,
 63409369, 63888049, 64144081, 64176121, 64272289, 64625521, 64850809, 64947481, 65108761,
 65302561, 65399569, 65431921, 65496649, 65626201, 65788321, 65885689, 65983129, 66373609,
 66601921, 66699889, 66765241, 66896041, 67092481, 67387681, 67551961, 67584841, 67749361,
 67782289, 67848169, 67947049, 68277169, 68376361, 68442529, 68674369, 68740681, 68773849,
 68840209, 69072721, 69172489, 69372241, 69772609, 69939769, 70040161, 70174129, 70341769,
 70375321, 70879561, 70946929, 71048041, 71081761, 71284249, 71351809, 71588521, 71690089,
 72267001, 72471169, 72607441, 72709729, 72880369, 72914521, 72982849, 73324969, 73496329,
 73633561, 73908409, 73942801, 74114881, 74356129, 74425129, 74459641, 74666881, 74770609,
 75047569, 75151561, 75290329, 75359761, 75498721, 75568249, 75672601, 75811849, 75916369,
 76020961, 76230361, 76335169, 76405081, 76510009, 76615009, 76755121, 77070841, 77141089,
 77492809, 77563249, 77774761, 77810041, 77986561, 78092569, 78127921, 78304801, 78517321,
 78552769, 78623689, 78978769, 79085449, 79619929, 79727041, 79798489, 79941481, 80120401,
 80335369, 80442961, 80478841, 80982001, 81018001, 81126049, 81198121, 81234169, 81522841,
 81739681, 81775849, 81884401, 82065481, 82210489, 82646281, 82864609, 82973881, 83302129,
 83411689, 83484769, 83740801, 83850649, 83923921, 84143929, 84290761, 84400969, 84621601,
 84695209, 84805681, 85026841, 85137529, 85359121, 85396081, 85692049, 86062729, 86136961,
 86174089, 86359849, 86694721, 86843761, 86918329, 87179569, 87254281, 87291649, 87403801,
 87815641, 87928129, 88190881, 88303609, 88416409, 88604569, 88717561, 88755241, 88943761,
 88981489, 89056969, 89094721, 89510521, 89548369, 89624089, 89737729, 89851441, 90079081,

```
90193009, 90459121, 90649441, 90878089, 90992521, 91145209, 91221601, 91910569, 92179201,
92409769, 92525161, 92602129, 92717641, 92756161, 92987449, 93103201, 93334921, 93644329,
93683041, 93876721, 94031809, 94458961, 94497841, 94731289, 94848121, 94926049, 95043001,
95394289, 95433361, 95667961, 95785369, 95863681, 96098809, 96255721, 96373489, 96609241,
96687889, 96805921, 97042201, 97160449, 97199881, 97436641, 97673689, 97752769, 98029801,
98148649, 98465929, 98585041, 98624761, 98823481, 98982601, 99341089, 99460729]
```

用暴力法也可以写出相应的代码，但是经过运算后发现**超时** (Time Limit Exceeded)。经过改动得到了判断法的代码，如下所示。

参考代码

时间: 750 ms 内存: 3516 kB

```
01. from math import sqrt
02.
03. tPrimes = {};
04. def isTPrime(x):
05.     global tPrimes;
06.     if x in tPrimes:
07.         return tPrimes[x];
08.     else:
09.         if int(int(sqrt(x)) ** 2) == x:
10.             root = int(sqrt(x));
11.             for i in range(2, int(sqrt(root) + 1)):
12.                 if root % i == 0:
13.                     tPrimes[x] = False;
14.                     return False;
15.             tPrimes[x] = True;
16.             return True;
17.
18. m, n = (int(i) for i in input().split());
19. for i in range(m):
20.     scores = [int(i) for i in input().split()];
21.     tPrimeScores = list(filter(isTPrime, scores));
22.     if len(tPrimeScores):
23.         print("%.2f" % (sum(tPrimeScores) / len(scores)));
24.     else:
25.         print(0);
```

D. 军备竞赛

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 18211 提交次数: 138
尝试人数: 68 通过人数: 50 通过率: 74%

描述

鸣人是木叶村的村长，最近在跟敌国进行军备竞赛，他手边有 n 份武器设计图，每张设计图有制作成（大于等于零）本且最多使用一次，可以选择花钱制作或是以同样的价钱卖给敌国，同时任意时刻敌国的武器不能比我国更多，鸣人的目标是在不负债的前提下武器种类比敌国越多越好。

输入

第一行为起始整数经费 p ，并且 $p \geq 0$ 。且要求任何时刻 p 不能小于 0。
第二行为 n 个整数，以空格分隔，并且每个整数 ≥ 0 。代表每张设计图的制作成本，同时也是卖价，最多用一次（无法又制作又卖）。

输出

一个整数，代表武器种类最多比敌国多多少。

样例输入 1

01.	10
02.	20 30 40

样例输出 1

01.	0
-----	---

解释：10 元不足以制作 20 元的武器，所以为 0，也不能先卖 50 元的^⑩，不能让敌国武器比木叶多。

样例输入 2

01.	10
02.	15 5

样例输出 2

01.	1
-----	---

解释：10 元可以制作 5 元的武器，木叶的武器比对手多一件。

样例输入 3

01.	40
02.	20 80 60 40

样例输出 3

01.	2
-----	---

解释：先制作 20 元的武器，再贩卖 80 元的武器，这时经费为 100，再制作 40、60 的武器，木叶的武器比对手多二件。

^⑩ 笔者注：原文如此。

思路

此题思路其实比较简单，把最便宜的留给自己，把最贵的买给敌人。设置变量数组 `fee` 存放所有武器的费用，然后对费用排序。先将所有经费用于制作最便宜的武器，并将制作过的武器费用从 `fee` 中移除。然后，如果有 2 件以上的武器剩余，且敌人武器数比自己少，则将贵的卖给敌人，用得到的钱继续制作武器。如果不能卖掉武器，则结束判断，输出比敌国多的武器。

参考代码

时间: 778 ms 内存: 3348 kB

```
01. p = int(input())
02. fee = [int(i) for i in input().split()];
03. fee.sort();
04. self, enemy = 0, 0;
05. while True:
06.     while fee and p >= fee[0]:
07.         p -= fee[0];
08.         fee.pop(0);
09.         self += 1;
10.     if len(fee) > 1 and enemy < self:
11.         p += fee[-1];
12.         fee.pop(-1);
13.         enemy += 1;
14.         continue;
15.     else:
16.         break;
17. print(self - enemy);
```

E. 图像的均值滤波

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 18188 提交次数: 155
尝试人数: 103 通过人数: 92 通过率: 89%

描述

背景知识:

1. 像素是指由图像的小方格组成的，这些小方块都有一个明确的位置和被分配的色彩数值，小方格颜色和位置就决定该图像所呈现出来的样子，对于灰度图而言，每个小方格即存放一个数值，即为像素值。
2. 数字图像在计算机中通常以二维数组（矩阵）的形式存放，其中每个像素值，即为该数组的值。
3. 图像的均值滤波操作指在图像上对目标像素给一个模板，该模板包括了其周围的临近像素以及自身，在这里我们考虑以目标像素为中心的 8 个像素，这时我们用模板中的全体像素的平均值来代替原来目标像素值，得到的结果即为图像均值滤波后的结果。

题目要求：实现图像的均值滤波。

给定一张输入图像，得到其均值滤波后的结果，得到的均值结果若是浮点数则向下取整。比如一个像素点均值为 2.5，则结果为 2。

边界点的像素只考虑在图像内部的。比如左上角的点邻居只有 3 个，分别是右边、下边、和右下。

输入

首先为 m 和 n 两个正整数，表示图像有 m 行 n 列。

接下来的 m 行每行有 n 个正整数代表输入图像的像素。

输出

m 行 n 列的二维数组，即经过均值滤波后的结果。

样例输入 1

01.	3 3
02.	2 3 2
03.	3 2 1
04.	1 1 1

样例输出 1

01.	2 2 2
02.	2 1 1
03.	1 1 1

解释:

考虑图像每个像素:

对于下标为 (0,0) 点，输入像素值为 2，输出计算为 $(2 + 3 + 3 + 2) / 4 = 2.5 \rightarrow 2$ 。

对于下标为 (0,1) 点，输入像素值为 3，输出计算为 $(2 + 3 + 2 + 3 + 2 + 1) / 6 = 2.17 \rightarrow 2$ 。

对于下标为 (1,1) 点，输入像素值为 2，输出计算为 $(2 + 3 + 2 + 3 + 2 + 1 + 1 + 1 + 1) / 9 = 1.78 \rightarrow 1$ 。

对于下标为 (2,2) 点，输入像素值为 1，输出计算为 $(2 + 1 + 1 + 1) / 4 = 1.25 \rightarrow 1$ 。

样例输入 2

```
01. 3 4
02. 2 3 2 4
03. 3 2 1 1
04. 1 1 1 1
```

样例输出 2

```
01. 2 2 2 2
02. 2 1 1 1
03. 1 1 1 1
```

思路

此题思路也比较简单，直接把每个点周围的点拉进来算一遍就行了。但是考虑到边界点的处理比较麻烦，因此定义了两个函数：

`safeCall` 函数用于判断给定的 (x, y) 值是否在边界范围内，若在范围内则返回对应值，若不在范围内则返回 `None`。

`safeAverage` 函数用于剔除所有 `None` 后求平均值。

参考代码

时间：650 ms 内存：3436 kB

```
01. def safeCall(matrix, x, y):
02.     if x >= 0 and x < len(matrix) and y >= 0 and y < len(matrix[0]):
03.         return matrix[x][y];
04.     else:
05.         return None;
06.
07. def safeAverage(lst):
08.     lst = list(filter(lambda x: x != None, lst));
09.     return sum(lst) // len(lst);
10.
11. matrix = ((-1, -1), (-1, 0), (-1, 1),
12.           (0, -1), (0, 0), (0, 1),
13.           (1, -1), (1, 0), (1, 1))
14.
15. m, n = (int(i) for i in input().split());
16. oldMatrix = [];
17. for i in range(m):
18.     oldMatrix.append([int(i) for i in input().split()]);
19. newMatrix = [[0 for i in range(n)] for i in range(m)];
20. for i in range(m):
21.     for j in range(n):
22.         lst = [safeCall(oldMatrix, i + k[0], j + k[1]) for k in matrix];
23.         newMatrix[i][j] = str(safeAverage(lst));
24.
25. print("\n".join([" ".join(j) for j in newMatrix]));
```

F. 统计套利策略

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 18177 提交次数: 46
 尝试人数: 13 通过人数: 1 通过率: 8%

描述

Mike 最近在课上学习了统计套利交易的简单思想, 他想自己编写程序来对市场上的品种 (股票、期货、外汇) 进行交易。在众多资产中, 他打算选取只两个品种来配对交易 (选定之后就无法更改)。试帮他找出选取哪两个品种可以获得最大的利润, 且计算该利润。

Mike 的策略:

1. 选定两种资产后, 每一个交易日, 他会计算之前所有日期两者价格的平均差值 (下面提示中的第一个公式), 作为他对两者价差的 “合理估计”。
2. 计算之前所有日期两者价差序列的标准差 (下面提示中的第二个公式) 作为对风险的 “合理估计”。
3. 在新的每一天, 如果选定两个品种的价差偏离历史平均一个标准差, Mike 预期会发生均值回归。故会相应地在两个品种分别持有 1 单位做空 (即卖出, 估计之后要跌) 和做多 (即买入, 估计之后要涨) 的仓位 (方向为假设发生均值回归会盈利的方向), 偏离历史平均两个标准差则持有 2 单位, 以此类推。

Mike 在第四天才开始交易, 以前三天的平均值和标准差作为初始值。

配对交易维基定义: Pairstrading is a market-neutral trading strategy that matches a long position with a short position in a pair of highly correlated instruments such as two stocks, exchange-traded funds (ETFs), currencies, commodities or options.

具体的例子: 比如下图中当两者价格偏离过大的时候, 就可以买入其中一者卖出另一者, 并期待其差值回归以获利。 (绿箭头代表买入, 红箭头代表卖出)

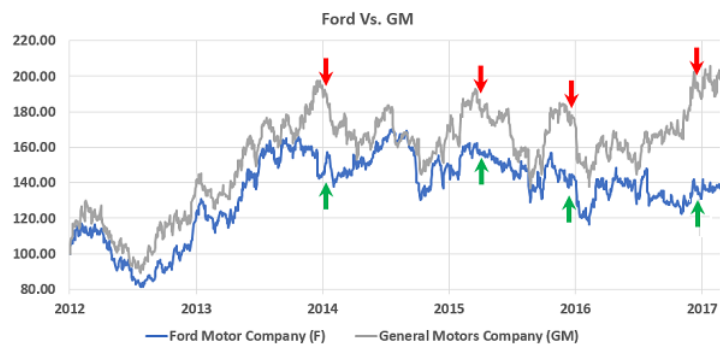


图 2 “统计套利策略” 题图

提示:

$$\text{mean}(P_A - P_B) = \frac{\sum_{i=1}^n (P_{A,i} - P_{B,i})}{n},$$

$$\text{std}(P_A - P_B) = \sqrt{\frac{\sum_{i=1}^n (P_{A,i} - P_{B,i} - \text{mean}(P_A - P_B))^2}{n}}.$$

参考伪代码:

```

01.  If  $P\_A(N + 1) - P\_B(N + 1) > \text{mean}(P\_A - P\_B) + k * \text{std}(P\_A - P\_B)$ 
02.  Then
03.      position(A) = k
04.      position(B) = -k

```

输入

第一行输入两个正整数，分别代表总共的品种数目 n ，和总共的天数 d 。

接着会输入 n 行，每行 d 个正整数，代表一个品种观测期间所有的价格数据。

输出

从第四天（含）起总收益最大的两个品种的序号（从小到大，序号从 1 开始，连续编号），以及最后的总收益（整数）。

（注：测试数据保证有唯一最优的套利对）

样例输入

01.	4 8							
02.	18	14	15	17	10	11	10	14
03.	19	15	18	18	15	11	15	16
04.	16	18	13	10	17	13	14	17
05.	20	12	11	16	14	17	11	15

样例输出

```

01. 1 3 38

```

解释:

样例计算过程分析:

此时选取第 1 和第 3 个资产可以取得最大利润。

资产 A 价格序列: 18 14 15 17 10 11 10 14。

资产 B 价格序列: 16 18 13 10 17 13 14 17。

差价定义为 $A - B$ ，则意味着向上偏离历史均值，A 高估，应做空，B 应做多。反之亦然。

- 第 4 天:
 - 历史平均差价: 0.00，历史方差: 2.83，今日差价: 7。
 - A 持有到明天单位收益 = $A(\text{Day}5) - A(\text{Day}4) = -7$ ，持有 A 做空手数: -2，收益: 14。
 - B 持到到明天单位收益 = $B(\text{Day}5) - B(\text{Day}4) = 7$ ，持有 B 做多手数: 2，收益: 14。
 - 总收益: 28，累积收益: 28。
- 第 5 天:
 - 历史平均差价: 1.75，历史方差: 3.90，今日差价: -7。
 - A 持有到明天单位收益 = $A(\text{Day}6) - A(\text{Day}5) = 1$ ，持有 A 手数: 2，收益: 2。
 - B 持到到明天单位收益 = $B(\text{Day}6) - B(\text{Day}5) = -4$ ，持有 B 手数: -2，收益: 8。
 - 总收益: 10，累积收益: 38。
- 第 6 天:
 - 历史平均差价: 0.00，历史方差: 4.94，今日差价: -2。

- A 持有到明天单位收益 = $A(\text{Day7}) - A(\text{Day6}) = -1$, 持有 A 手数: 0, 收益: 0。
- B 持有到明天单位收益 = $B(\text{Day7}) - B(\text{Day6}) = 1$, 持有 B 手数: 0, 收益: 0。
- 总收益: 0, 累积收益: 38。
- 第 7 天:
 - 历史平均差价: -0.33, 历史方差: 4.57, 今日差价: -4。
 - A 持有到明天单位收益 = $A(\text{Day8}) - A(\text{Day7}) = 4$, 持有 A 手数: 0, 收益: 0。
 - B 持有到明天单位收益 = $B(\text{Day8}) - B(\text{Day7}) = 3$, 持有 B 手数: 0, 收益: 0。
 - 总收益: 0, 累积收益: 38。

思路

这题我没做出来……把错误代码扔到这里大家自己看吧……

参考代码

```
01. from math import sqrt
02.
03. def MEAN(A, B):
04.     return sum([A[i] - B[i] for i in range(len(A))]) / len(A);
05.
06. def STD(A, B, mean):
07.     return sqrt(sum([(A[i] - B[i] - mean) ** 2 for i in range(len(A))]) / len(A));
08.
09. n, d = [int(i) for i in input().split()];
10. prices = [];
11. for i in range(n):
12.     prices.append([int(i) for i in input().split()]);
13.
14. maxProfit = [0, 0, 0];
15. for i in range(n):
16.     for j in range(i + 1, n):
17.         profit = 0;
18.         for k in range(3, d - 1):
19.             A, B = prices[i][:k], prices[j][:k];
20.             mean = MEAN(A, B);
21.             std = STD(A, B, mean);
22.             diff = prices[i][k] - prices[j][k];
23.             K = int((diff - mean) / std);
24.             profit += (prices[i][k] - prices[i][k + 1]) * K - (prices[j][k] - prices[j][k + 1]) *
                K;
25.             print(profit);
26.             if profit > maxProfit[2]:
27.                 maxProfit = [i + 1, j + 1, profit];
28. print(" ".join([str(i) for i in maxProfit]));
```

2019 级期末考试

A. 这一天星期几

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 19944 提交次数: 415

尝试人数: 135 通过人数: 118 通过率: 87%

描述

在日常生活中，计算某一个具体的日期是星期几，往往需要去翻阅日历。请你帮助更快地计算出每个日期在一星期是第几天。

参考:

蔡勒公式 (Zeller's congruence)，是一种计算任何一日属一星期中哪一日的算法，由德国数学家克里斯提安·蔡勒推算出来，可以计算 1582 年 10 月 15 日之后的情况。

$$w = \left(y + \left\lfloor \frac{y}{4} \right\rfloor + \left\lfloor \frac{c}{4} \right\rfloor - 2c + \left\lfloor \frac{26(m+1)}{10} \right\rfloor + d - 1 \right) \bmod 7.$$

公式都是基于公历的置闰规则来考虑。

公式中的符号含义如下:

w : 星期 (计算所得的数值对应的星期: 0-Sunday; 1-Monday; 2-Tuesday; 3-Wednesday; 4-Thursday; 5-Friday; 6-Saturday)。

c : 年份前两位数。

y : 年份后两位数。

m : 月 (m 的取值范围为 3 至 14，即在蔡勒公式中，某年的 1、2 月要看作上一年的 13、14 月来计算，比如 2003 年 1 月 1 日要看作 2002 年的 13 月 1 日来计算)。

d : 日。

$\lfloor \cdot \rfloor$: 称作高斯符号，代表向下取整，即，取不大于原数的最大整数。

\bmod : 同余 (这里代表括号里的答案除以 7 后的余数)。

输入

第一行 1 个整数 n ，代表输入日期的个数。

接下来 n 行分别为 n 个以 8 位数字表示的日期，如 20190101。

保证所有输入都在蔡勒公式的计算范围之内。

输出

共 n 行，每行为该日期对应的 weekday 名称 (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday)。

样例输入

01.	15
02.	19850706
03.	19710710

04.	20041125
05.	20141220
06.	19841128
07.	19760429
08.	19931102
09.	19951002
10.	20161006
11.	20151226
12.	19790715
13.	20080410
14.	20091104
15.	19910621
16.	19891012

样例输出

01.	Saturday
02.	Saturday
03.	Thursday
04.	Saturday
05.	Wednesday
06.	Thursday
07.	Tuesday
08.	Monday
09.	Thursday
10.	Saturday
11.	Sunday
12.	Thursday
13.	Wednesday
14.	Friday
15.	Thursday

思路

我觉得 2019 年的期末简直是在放水……

这题不太需要什么思路，直接按题目给出的公式计算即可。要注意的是如果 $m < 3$ 不仅需要令 $m = m + 12$ ，还要把年份-1（另外需要考虑世纪变化），否则是无法通过的（我就是在这里卡住的）。

（补充：偶然翻知乎看到一个答案，才想起来 python 是有自带的日期库的……随手重新写了一段代码，见参考代码 2）^①

参考代码 1

时间： 779 ms 内存： 3640 kB

01.	weekName = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];
02.	
03.	n = int(input());

^① 参考资料：Super Mario. 在信息学竞赛中，你见过哪些“这都能 AC!?” 的题? - Super Mario 的回答 [EB/OL]. 知乎. (2019-11-24) [2019-12-30]. <https://www.zhihu.com/question/343126456/answer/881280919>.

```
04.     for i in range(n):
05.         i = input();
06.         c, y, m, d = (int(i[x:x + 2]) for x in range(0, 8, 2));
07.         if m < 3:
08.             m += 12;
09.             if y > 0:
10.                 y -= 1;
11.             else:
12.                 y = 99;
13.                 c -= 1;
14.         w = (y + y // 4 + c // 4 - 2 * c + 26 * (m + 1) // 10 + d - 1) % 7;
15.         print(weekName[w]);
```

参考代码 2

时间: 920 ms 内存: 4084 kB

```
01.     import datetime
02.
03.     n = int(input());
04.     for i in range(n):
05.         i = input();
06.         c, y, m, d = (int(i[x:x + 2]) for x in range(0, 8, 2));
07.         print(datetime.date(c * 100 + y, m, d).strftime("%A"));
```

B. 提取实体

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 19949 提交次数: 354
尝试人数: 116 通过人数: 74 通过率: 64%

描述

一个句子里面一些特殊的单词被称作实体，实体是存在于现实世界中并且可以与其他物体区分开来的物体，如“John has an apple.”这句话中，“John”和“apple”都是实体。
现在有很多个人工标注好实体的英文文档，每篇文档有很多个句子，每个句子中，每个实体的每单词都添加了“###”前缀，并且添加了“###”后缀，表明两个“###”之间的部分是实体或者是实体的一部分。例如：

- 1) 两个“###”之间是实体：
“###John### has an ###apple###”中有两个实体，是 John 和 apple。
- 2) 两个“###”之间是实体的一部分，即是，连续的几个被“###”前后包裹的单词被认为是同一个实体：
“###Shelley### ###Berkley### , a Democratic representative”中有一个实体，是 Shelley Berkley。
“###Dominic### ###J.### ###Baranello### , an enduring power in Democratic Party”中有一个实体，是 Dominic J. Baranello。

请你帮助统计每篇文档里有多少个实体，暂时不考虑句子间的实体有重复的情况。

输入

第一行为 1 个整数 n ，代表文档里的句子数目。
接下来 n 行，每行代表一个英文句子，每个句子有多少单词是未知的，词与词之间用空格分隔。

输出

1 个整数，代表该篇文档里的实体数目。

样例输入 1

01.	1
02.	###John### has an ###apple### .

样例输出 1

01.	2
-----	---

解释：文档中只有一个句子，该句子中有两个实体“John”和“apple”，所以输出是 2。

样例输入 2

01.	1
02.	###Shelley### ###Berkley### , a Democratic representative of Nevada Mrs. ###Babbitt### 's daughter lives in ###Las### ###Vegas### testified about the case in July at a Congressional hearing into the recovery of art stolen during World War II .

样例输出 2

01.	3
-----	---

解释：文档中仍然只有一个句子，但是现在有三个实体，“Shelley Berkley”，“Babitt”和“Las

Vegas”。

思路

看到这个题我第一反应是正则表达式匹配……但是考虑到正则表达式需要一定的基础，因此我换了一种简单的思路。

由于原题中提到“连续的几个被‘###’前后包裹的单词被认为是同一个实体”，因此可以直接把所有“### ###”删掉，然后计算所有“###”的数目再除以 2 即可。

参考代码

时间: 694 ms 内存: 3800 kB

```
01. n = int(input());
02. text = "\n".join([input() for i in range(n)]);
03. print(len(text.replace("### ###", "").split("###")) // 2);
```

C. 图的拉普拉斯矩阵

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 19943 ^⑫ 提交次数: 165
尝试人数: 110 通过人数: 94 通过率: 85%

描述

在图论中，度数矩阵是一个对角矩阵，其中包含的信息为的每一个顶点的度数，也就是说，每个顶点相邻的边数。邻接矩阵是图的一种常用存储方式。如果一个图一共有编号为 $0, 1, 2, \dots, n-1$ 的 n 个节点，那么邻接矩阵 A 的大小为 $n \times n$ ，对其中任一元素 A_{ij} ，如果节点 i, j 直接有边，那么 $A_{ij} = 1$ ；否则 $A_{ij} = 0$ 。

将度数矩阵与邻接矩阵逐位相减，可以求得图的拉普拉斯矩阵。具体可见下图示意。

$$L := D - A,$$
$$L := D - A$$

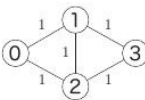

$$D = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad L = \begin{pmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}$$

图 3 “提取实体” 题图

现给出一个图中的所有边的信息，需要你输出该图的拉普拉斯矩阵。

输入

第一行 2 个整数，代表该图的顶点数 n 和边数 m 。

接下 m 行，每行为空格分隔的 2 个整数 a 和 b ，代表顶点 a 和顶点 b 之间有一条无向边相连， a 和 b 均为大小范围在 0 到 $n-1$ 之间的整数。输入保证每条无向边仅出现一次（如 1 2 和 2 1 是同一条边，并不会在数据中同时出现）。

输出

共 n 行，每行为以空格分隔的 n 个整数，代表该图的拉普拉斯矩阵。

样例输入

01.	4 5
02.	2 1
03.	1 3
04.	2 3
05.	0 1
06.	0 2

样例输出

01.	2 -1 -1 0
02.	-1 3 -1 -1
03.	-1 -1 3 -1
04.	0 -1 -1 2

^⑫ 笔者注：此链接无法打开，似乎是老师没有公开此题作为练习题。

思路

这道题并没有开放练习，因此我也不知道我做得对不对，讲一下思路。

先设置两个空矩阵 D 、 A ，矩阵元素均设置为 0；然后依次读取各条边，把 D 中对应的数+1、 A 中对应的数设为 1。最后直接输出两个矩阵的差即可。

参考代码

时间: ??? ms 内存: ???? kB

```
01. n, m = [int(i) for i in input().split()];
02. D, A = ([[0 for i in range(n)] for j in range(n)] for k in range(2));
03. for i in range(m):
04.     x, y = [int(i) for i in input().split()];
05.     D[x][x] += 1;
06.     D[y][y] += 1;
07.     A[x][y] = 1;
08.     A[y][x] = 1;
09.
10. print("\n".join([" ".join([str(D[j][i] - A[j][i]) for i in range(n)]) for j in range(n)]));
```

D. 二维矩阵上的卷积运算

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 19942 提交次数: 134
尝试人数: 82 通过人数: 73 通过率: 89%

描述

二维矩阵上的卷积，是卷积神经网络中经常需要进行的一种运算。该运算在输入的二维矩阵上滑动不同的卷积核，并在每一个滑动的位置上将卷积核与输入图像对应位置的元素进行相乘并逐个求和的运算。下图很好地说明了运算的结果矩阵的每一个位置是如何计算得来的。本题中在行列方向的滑动均以 1 为步长进行，且输入输出均为整数。

如对第 1 行第 1 列， $12 = 3 \times 0 + 3 \times 1 + 2 \times 2 + 0 \times 2 + 0 \times 2 + 1 \times 0 + 3 \times 0 + 1 \times 1 + 2 \times 2$ 。

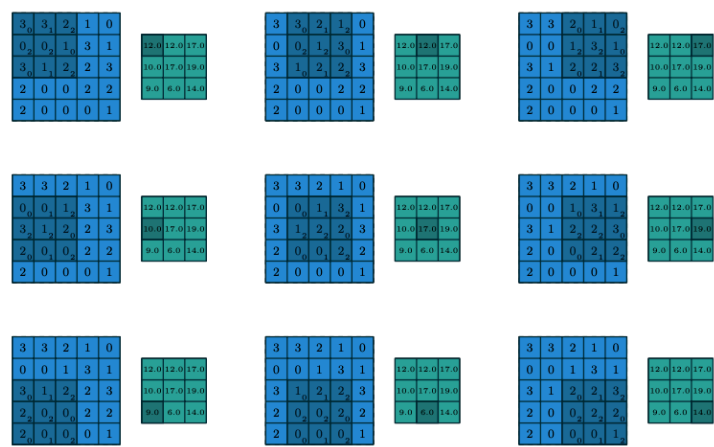


图 4 “二维矩阵上的卷积运算” 题图

输入

第一行 4 个整数，分别代表二维矩阵的行数和列数 m 和 n 以及卷积核的行数和列数 p 和 q ($1 \leq p \leq m$, $1 \leq q \leq n$)。

接下来 m 行，每行为空格分隔的 n 个整数，代表二维数组中每行的数据。

接下来 p 行，每行为空格分隔的 q 个整数，代表卷积核中每行的数据。

输出

易知， $p \times q$ 的卷积核在 $m \times n$ 的矩阵上以 1 为步长滑动，横向一共有 $m + 1 - p$ 个位置，纵向一共有 $n + 1 - q$ 个位置。

因此输出共 $m + 1 - p$ 行，每行为以空格分隔的 $n + 1 - q$ 个整数，代表卷积运算后的结果。

样例输入 1

01.	5 5 3 3
02.	3 3 2 1 0
03.	0 0 1 3 1
04.	3 1 2 2 3
05.	2 0 0 2 2
06.	2 0 0 0 1
07.	0 1 2
08.	2 2 0

09.	0 1 2
-----	-------

样例输出 1

01.	12 12 17
02.	10 17 19
03.	9 6 14

样例输入 2

01.	5 4 4 4
02.	10 -8 6 9
03.	7 -9 1 0
04.	1 -9 2 -5
05.	-3 6 -1 2
06.	-10 -2 -1 -2
07.	-9 -1 -6 10
08.	3 -2 -5 9
09.	-10 -3 -10 7
10.	3 -2 -7 0

样例输出 2

01.	-46
02.	-77

思路

先设置两个矩阵 a , b 分别存放二维数组和卷积核, 然后依次遍历 $m+1-p$ 行和 $n+1-q$ 列上的数字, 对于每个位置上的数字也要遍历二维数组和卷积核对应位置的乘积和, 然后输出。

参考代码

时间: 761 ms 内存: 3640 kB

```

01. m, n, p, q = (int(i) for i in input().split());
02. a = [[int(i) for i in input().split()] for j in range(m)];
03. b = [[int(i) for i in input().split()] for j in range(p)];
04.
05. result = [[0 for i in range(n + 1 - q)] for j in range(m + 1 - p)];
06. for i in range(m + 1 - p):
07.     for j in range(n + 1 - q):
08.         c = sum(sum([a[x + i][y + j] * b[x][y] for y in range(q)]) for x in range(p));
09.         print(c, end = " " if j < n - q else "\n");

```

E. 买学区房

总时间限制: 1000 ms

内存限制: 65536 kB

全局题号: 19963

提交次数: 145

尝试人数: 71

通过人数: 50

通过率: 70%

描述

小明同学的家长为了让小明同学接受更好的教育，最近在考虑买某重点中学附近的房子，已知他们买房子要考虑两个因素：房子离学校的距离，以及房子的价格。现在他们有一系列备选的房子，已知这些房子距离学校的 x 方向距离和 y 方向距离，以及每栋房子的价格。小明的家长认为，只有同时满足了以下两个条件的房子 H 买了才是不亏的：

1. 小明的家长比较精打细算，所以房子的性价比大于所用备选房子性价比的中位数（定义见下图）^⑬。
2. 小明的家长想攒钱，所以房子的价格小于所有备选房子价格的中位数。

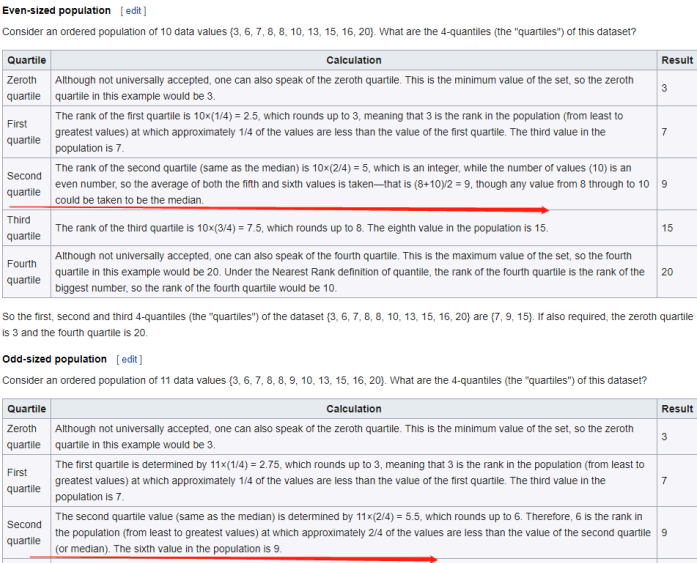


图 5 “买学区房” 题图

注：性价比 = 房子和学校之间的交通距离 / 房子的价格。

（由于该城市的街道布局接近方格状，且从学校到房子无法穿墙而过，房子 H 去学校的交通距离定义为，房子 H 距离学校的 x 方向距离和 y 方向距离的和）。

现在需要你来帮小明的家长判断，一系列备选房子里，值得买房子有多少栋。

提示：求中位数，要先进行数据的排序（从小到大），然后计算中位数的序号，分数据为奇数与偶数两种来求。排序时，相同的数字不能省略。

如果总数个数是奇数的话，按从小到大的顺序，取中间的那个数。

如果总数个数是偶数的话，按从小到大的顺序，取中间那两个数的平均数。

输入

- 第 1 行为 1 个整数， n ，代表备选房子的数目。
- 第 2 行为 n 个房子离学校的 x, y 距离对，如 “ (x, y) ”，距离均为整数。
- 第 3 行为 n 个整数，代表每个房子的价格。

^⑬ 笔者注：原文如此。此处的图片有误导，本题在验证时选取的中位数计算方法应该按照“提示”中所述，选取中间两数平均数。

输出

一个整数，代表 n 个房子中值得买房子数目。

样例输入 1

```
01. 5
02. (100,200) (50,50) (100,300) (150,50) (50,50)
03. 100 300 200 400 500
```

样例输出 1

```
01. 2
```

说明：共有 5 个房子备选，离学校的交通距离分别是 $100 + 200 = 300$ ， $50 + 50 = 100$ ， $100 + 300 = 400$ ， $150 + 50 = 200$ ， $50 + 50 = 100$ 。这些房子的性价比依次为 $300 / 100 = 3$ ， $100 / 300 = 1 / 3$ ， $400 / 200 = 2$ ， $200 / 400 = 0.5$ ， $100 / 500 = 0.2$ ，中位数是 0.5，满足第 1 个条件的只有第 1 个和第 3 个房子。这些房子的价格中位数是 300，因此满足第 2 个条件的只有第 1 个和第 3 个房子。所以同时满足两个条件的有第 1 个和第 3 个房子，输出 2。

样例输入 2

```
01. 3
02. (10,90) (20,180) (30,270)
03. 100 200 300
```

样例输出 2

```
01. 0
```

说明：共有三个房子备选，离学校的交通距离分别是 $10 + 90 = 100$ ， $20 + 180 = 200$ ， $30 + 270 = 300$ 。这些房子性价比依次为 $100 / 100 = 1$ ， $200 / 200 = 1$ ， $300 / 300 = 1$ ，中位数是 1，没有房子满足第 1 个条件，因此不用考虑第二个条件，输出一定是 0。

思路

首先定义求中位数函数 `getMedian`。这里使用了偷懒的办法，先将列表排序，然后直接返回最中间两个数的平均值，无论列表长度是奇数还是偶数（因为即使列表长度是奇数，中位数也是中间两数之平均值）。

然后就是常规操作了，先输入距离（这里距离求算有些麻烦）和价格，然后求性价比，再筛选出所有性价比高于中位数和价格低于中位数的房子，最后计数并输出。

参考代码

时间：810 ms 内存：4188 kB

```
01. def getMedian(l):
02.     l = sorted(l);
03.     return (l[len(l) // 2] + l[(len(l) - 1) // 2]) / 2;
04.
05. n = int(input());
06. distance = [sum([int(j) for j in i[1:-1].split(",")]) for i in input().split()];
07. price = [int(i) for i in input().split()];
08. ratio = [distance[i] / price[i] for i in range(n)];
09.
```

```
10. ratioM = getMedian(ratio);
11. priceM = getMedian(price);
12. count = [(True if (ratio[i] > ratioM and price[i] < priceM) else False) for i in range(n)];
13.
14. print(count.count(True));
```

F. 因材施教

总时间限制: 1000 ms 内存限制: 65536 kB 全局题号: 19948 提交次数: 47

尝试人数: 37 通过人数: 26 通过率: 70%

描述

有一所魔法高校招入一批学生，为了贯彻因材施教的理念，学校打算根据他们的魔法等级进行分班教育。在确定班级数目的情况下，班级内学生的差异要尽可能的小，也就是各个班级内学生的魔法等级要尽可能的接近。

例如：现在有 $n = 7$ 位学生，他们的魔法等级分别为 $r = [2, 7, 9, 9, 16, 28, 45]$ ，我们要将他们分配到 $m = 3$ 个班级，如果按照 $([2, 7], [9, 9], [16, 28, 45])$ 的方式分班，则他们的总体差异为 $d = (7 - 2) + (9 - 9) + (45 - 16) = 34$ 。

输入

第一行为两个整数：学生人数 n 和班级数目 m ， $1 \leq m \leq n \leq 10^5$ 。

第二行为 n 个整数：每位学生的魔法等级 r_i ， $1 \leq r_i \leq 10^9$ 。

输出

一个整数：学生的最小总体差异 d 。

样例输入 1

```
01. 7 3
02. 2 7 9 9 16 28 45
```

样例输出 1

```
01. 14
```

说明：最小总体差异的分班方式为 $([2, 7, 9, 9, 16], [28], [45])$ 。

样例输入 2

```
01. 15 9
02. 90 73 116 47 400 212 401 244 13 372 248 56 194 482 177
```

样例输出 2

```
01. 65
```

说明：最小总体差异的分班方式为 $([13], [47, 56, 73, 90], [116], [177, 194], [212], [244, 248], [372], [400, 401], [482])$ 。

思路

这个题乍一看没看懂，再一看看懂了但是没思路，想了一会觉得可以模拟分班方式，最后算了一下发现不用这么麻烦。

先讲原理：由于总体差异是每个班级中最高级和最低级的级别之差的总和，因此应使每个班级的差尽可能小。注意到，如果某班级仅有一个人，那么该班级的差异为 0。因此应该把那些和别人相差较大的人单独分班，转换成程序语言就是：先把所有能力值排序，然后计算相邻两项之差。这时注意到 n 位学生分到 m 个班级中会产生 $n - m$ 个差异，因此把相邻两项之差再次排序，选取前 $n - m$ 个加和即为所求。

由于本题仅要求出最小总体差异，因此到这一步就可以结束了。如果要求相应的分班方式，也可采用同一

思想。

另外，本题的标题标注了 “greedy”，意为 “贪心算法”。但我个人觉得我使用的方法并没有用到贪心算法。

参考代码

时间: 3777 ms 内存: 26796 kB

```
01. m, n = [int(i) for i in input().split()];
02. score = sorted([int(i) for i in input().split()]);
03. print(sum(sorted([score[i] - score[i - 1] for i in range(1, m)][:m - n])));
```


后记

老子曾经说过，一篇文章有前言就应该有后记。

这篇资料是我为了一个女孩写的。遗憾的是，我并没有能够打动她，于是我选择切断了和她的联系。即便如此，我还是把这篇资料完成了。

因为写的时候我是按照“入门级资料”的思想编写的，因此我把题目要求都放了上来，看上去显得比较完备。但是这也导致整篇资料非常冗长。不过我觉得应该不会有人把它一页一页打印下来吧……既然是电子化阅读，多一点也无所谓了，反正也占不了多少空间。

今年的北京至今已经下了两场大雪。什么时候才能有人陪我一起看雪呢？或许这个问题只有上天才知道答案吧。

那么，有缘再见。

张祥伟

2019 年 12 月 28 日于燕园

